

# NSScriptCommand Class Reference

# Contents

## **NSScriptCommand Class Reference** 4

Overview 4

Adopted Protocols 5

Tasks 5

    Initializing a Script Command 5

    Getting the Current Command 5

    Getting the Apple Event 5

    Executing the Command 6

    Accessing Receivers 6

    Accessing Arguments 6

    Accessing the Direct Parameter 6

    Getting Command Information 7

    Handling Script Execution Errors 7

    Suspending and Resuming Commands 8

Class Methods 8

    currentCommand 8

Instance Methods 9

    appleEvent 9

    arguments 9

    commandDescription 9

    directParameter 10

    evaluatedArguments 11

    evaluatedReceivers 11

    executeCommand 12

    initWithCommandDescription: 12

    isWellFormed 13

    performDefaultImplementation 13

    receiversSpecifier 14

    resumeExecutionWithResult: 14

    scriptErrorExpectedTypeDescriptor 15

    scriptErrorNumber 16

    scriptErrorOffendingObjectDescriptor 16

    scriptErrorString 17

    setArguments: 17

- setDirectParameter: 18
- setReceiversSpecifier: 18
- setScriptErrorExpectedTypeDescriptor: 19
- setScriptErrorNumber: 20
- setScriptErrorOffendingObjectDescriptor: 20
- setScriptErrorString: 21
- suspendExecution 22

Constants 22

- NSScriptCommand—General Command Execution Errors 22

**Document Revision History** 25

# NSScriptCommand Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSCoding NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/Foundation.framework
<b>Availability</b>	Available in OS X v10.0 and later.
<b>Companion guide</b>	Cocoa Scripting Guide
<b>Declared in</b>	NSScriptCommand.h
<b>Related sample code</b>	SimpleScriptingPlugin SimpleScriptingVerbs Sketch Sketch+Accessibility

---

## Overview

An instance of `NSScriptCommand` represents a scripting statement, such as `set word 5 of the front document to word 1 of the second document`, and contains the information needed to perform the operation specified by the statement.

When an Apple event reaches a Cocoa application, Cocoa's built-in scripting support transforms it into a script command (that is, an instance of `NSScriptCommand` or one of the subclasses provided by Cocoa scripting or by your application) and executes the command in the context of the application. Executing a command means either invoking the selector associated with the command on the object or objects designated to receive the command, or having the command perform its default implementation method ([performDefaultImplementation](#) (page 13)).

Your application most likely calls methods of `NSScriptCommand` to extract the command arguments. You do this either in the `performDefaultImplementation` method of a command subclass you have created, or in an object method designated as the selector to handle a particular command.

As part of Cocoa's standard scripting implementation, `NSScriptCommand` and its subclasses can handle the default command set for AppleScript's Standard suite for most applications without any subclassing. The Standard suite includes commands such as `copy`, `count`, `create`, `delete`, `exists`, and `move`, as well as common object classes such as `application`, `document`, and `window`.

For more information on working with script commands, see *Script Commands* in *Cocoa Scripting Guide*.

## Adopted Protocols

### NSCoding

- `encodeWithCoder:`
- `initWithCoder:`

## Tasks

### Initializing a Script Command

---

- `initWithCommandDescription:` (page 12)  
Returns an a script command object initialized from the passed command description.

### Getting the Current Command

---

- + `currentCommand` (page 8)  
If a command is being executed in the current thread by Cocoa scripting's built-in Apple event handling, return the command.

### Getting the Apple Event

---

- `appleEvent` (page 9)  
If the receiver was constructed by Cocoa scripting's built-in Apple event handling, returns the Apple event descriptor from which it was constructed.

## Executing the Command

---

- [executeCommand](#) (page 12)  
Executes the command if it is valid and returns the result, if any.
- [performDefaultImplementation](#) (page 13)  
Overridden by subclasses to provide a default implementation for the command represented by the receiver.

## Accessing Receivers

---

- [evaluatedReceivers](#) (page 11)  
Returns the object or objects to which the command is to be sent (called both the “receivers” or “targets” of script commands).
- [receiversSpecifier](#) (page 14)  
Returns the object specifier that, when evaluated, yields the receiver or receivers of the command.
- [setReceiversSpecifier:](#) (page 18)  
Sets the object specifier to `receiversSpec` that, when evaluated, indicates the receiver or receivers of the command.

## Accessing Arguments

---

- [arguments](#) (page 9)  
Returns the arguments of the command.
- [evaluatedArguments](#) (page 11)  
Returns a dictionary containing the arguments of the command, evaluated from object specifiers to objects if necessary. The keys in the dictionary are the argument names.
- [setArguments:](#) (page 17)  
Sets the arguments of the command to `args`.

## Accessing the Direct Parameter

---

- [directParameter](#) (page 10)  
Returns the object that corresponds to the direct parameter of the Apple event from which the receiver derives.

- [setDirectParameter:](#) (page 18)  
Sets the object that corresponds to the direct parameter of the Apple event from which the receiver derives.

## Getting Command Information

---

- [commandDescription](#) (page 9)  
Returns the command description for the command.
- [isWellFormed](#) (page 13)  
Returns a Boolean value indicating whether the receiver is well formed according to its command description.

## Handling Script Execution Errors

---

- [scriptErrorExpectedTypeDescriptor](#) (page 15)  
Returns the type descriptor that was put in the reply Apple event if the sender requested a reply, execution of the receiver completed, and an error number was set.
- [scriptErrorNumber](#) (page 16)  
Returns the script error number, if any, associated with execution of the command.
- [scriptErrorOffendingObjectDescriptor](#) (page 16)  
Returns the object descriptor that was put in the reply Apple event if the sender requested a reply, execution of the receiver completed, and an error number was set.
- [scriptErrorString](#) (page 17)  
Returns the script error string, if any, associated with execution of the command.
- [setScriptErrorExpectedTypeDescriptor:](#) (page 19)  
Sets a descriptor for the expected type that will be put in the reply Apple event if the sender requested a reply, execution of the receiver completes, and an error number was set.
- [setScriptErrorOffendingObjectDescriptor:](#) (page 20)  
Sets a descriptor for an object that will be put in the reply Apple event if the sender requested a reply, execution of the receiver completes, and an error number was set.
- [setScriptErrorNumber:](#) (page 20)  
Sets a script error number that is associated with the execution of the command and is returned in the reply Apple event, if a reply was requested by the sender.
- [setScriptErrorString:](#) (page 21)  
Sets a script error string that is associated with execution of the command.

## Suspending and Resuming Commands

---

- [suspendExecution](#) (page 22)  
Suspends the execution of the receiver.
- [resumeExecutionWithResult:](#) (page 14)  
If a successful, unmatched, invocation of [suspendExecution](#) (page 22) has been made, resume the execution of the command.

## Class Methods

### **currentCommand**

---

*If a command is being executed in the current thread by Cocoa scripting's built-in Apple event handling, return the command.*

+ (NSScriptCommand \*)currentCommand

#### **Discussion**

A command is being executed in the current thread by Cocoa scripting's built-in Apple event handling if an instance of `NSScriptCommand` is handling an [executeCommand](#) (page 12) message at this instant as the result of the dispatch of an Apple event. Returns `nil` otherwise. [setScriptErrorNumber:](#) (page 20) and [setScriptErrorString:](#) (page 21) messages sent to the returned command object will affect the reply event sent to the sender of the event from which the command was constructed, if the sender has requested a reply.

A suspended command is not considered the current command. If a command is suspended and no other command is being executed in the current thread, `currentCommand` returns `nil`.

#### **Availability**

Available in OS X v10.3 and later.

#### **Related Sample Code**

Sketch

Sketch+Accessibility

#### **Declared in**

`NSScriptCommand.h`

## Instance Methods

### appleEvent

---

If the receiver was constructed by Cocoa scripting's built-in Apple event handling, returns the Apple event descriptor from which it was constructed.

– (NSAppleEventDescriptor \*)appleEvent

#### Discussion

The effects of mutating or retaining this descriptor are undefined, although it may be copied.

#### Availability

Available in OS X v10.3 and later.

#### Declared in

NSScriptCommand.h

### arguments

---

Returns the arguments of the command.

– (NSDictionary \*)arguments

#### Discussion

If there are no arguments, returns an empty `NSDictionary` object. When you subclass `NSScriptCommand` or one of its subclasses, you rarely call this method because it returns the arguments directly, without evaluating any arguments that are object specifiers. If any of a command's arguments may be object specifiers, which is generally the case, call [evaluatedArguments](#) (page 11) instead.

#### Availability

Available in OS X v10.0 and later.

#### See Also

– [setArguments:](#) (page 17)

#### Declared in

NSScriptCommand.h

### commandDescription

---

Returns the command description for the command.

– (NSScriptCommandDescription \*)commandDescription

### Discussion

Once a command is created, its command description is immutable.

### Availability

Available in OS X v10.0 and later.

### See Also

– [isWellFormed](#) (page 13)

### Declared in

NSScriptCommand.h

---

## directParameter

*Returns the object that corresponds to the direct parameter of the Apple event from which the receiver derives.*

– (id)directParameter

### Return Value

An object. Returns `nil` if the received Apple event doesn't contain a direct parameter.

### Discussion

For example, the direct parameter of a `print documents` Apple event contains a list of documents. This method may return the same object or objects returned by [receiversSpecifier](#) (page 14).

### Availability

Available in OS X v10.0 and later.

### See Also

– [setDirectParameter:](#) (page 18)

### Related Sample Code

SimpleScriptingPlugin

SimpleScriptingVerbs

### Declared in

NSScriptCommand.h

## evaluatedArguments

---

Returns a dictionary containing the arguments of the command, evaluated from object specifiers to objects if necessary. The keys in the dictionary are the argument names.

– (NSDictionary \*)evaluatedArguments

### Discussion

Arguments initially can be either a normal object or an object specifier such as `word 5` (represented as an instance of an `NSScriptObjectSpecifier` subclass). If arguments are object specifiers, the receiver evaluates them before using the referenced objects. Returns `nil` if the command is not well formed. Also returns `nil` if an object specifier does not evaluate to an object or if there is no type defined for the argument in the command description.

### Availability

Available in OS X v10.0 and later.

### See Also

- [isWellFormed](#) (page 13)
- [arguments](#) (page 9)
- [setArguments:](#) (page 17)

**Related Sample Code**  
[SimpleScriptingPlugin](#)  
[SimpleScriptingVerbs](#)

### Declared in

`NSScriptCommand.h`

## evaluatedReceivers

---

Returns the object or objects to which the command is to be sent (called both the “receivers” or “targets” of script commands).

– (id)evaluatedReceivers

### Discussion

It evaluates receivers, which are always object specifiers, to a proper object. If the command does not specify a receiver, or if the receiver doesn’t accept the command, it returns `nil`.

### Availability

Available in OS X v10.0 and later.

### See Also

- [receiversSpecifier](#) (page 14)
- [setReceiversSpecifier:](#) (page 18)

### Declared in

NSScriptCommand.h

## executeCommand

---

*Executes the command if it is valid and returns the result, if any.*

– (id)executeCommand

### Discussion

Before this method executes the command (through `NSInvocation` mechanisms), it evaluates all object specifiers involved in the command, validates that the receivers can actually handle the command, and verifies that the types of any arguments that were initially object specifiers are valid.

You shouldn't have to override this method. If the command's receivers want to handle the command themselves, this method invokes their defined handler. Otherwise, it invokes [performDefaultImplementation](#) (page 13).

### Availability

Available in OS X v10.0 and later.

### See Also

- [evaluatedArguments](#) (page 11)
- [evaluatedReceivers](#) (page 11)

### Declared in

NSScriptCommand.h

## initWithCommandDescription:

---

*Returns an a script command object initialized from the passed command description.*

– (id)initWithCommandDescription:(NSScriptCommandDescription \*)commandDesc

### Parameters

commandDesc

A command description for the command to be created.

### Return Value

A newly initialized instance of `NSScriptCommand` or a subclass.

### Discussion

To make this command object usable, you must set its receiving objects and arguments (if any) after invoking this method.

### Availability

Available in OS X v10.0 and later.

### See Also

- [setArguments:](#) (page 17)
- [setReceiversSpecifier:](#) (page 18)

### Declared in

`NSScriptCommand.h`

---

## **isWellFormed**

*Returns a Boolean value indicating whether the receiver is well formed according to its command description.*

- (BOOL)isWellFormed

### Discussion

The method ensures that there is a description of the command and that the number of arguments and the types of non-specifier arguments conform to the command description.

### Availability

Available in OS X v10.0 and later.

### See Also

- [commandDescription](#) (page 9)

### Declared in

`NSScriptCommand.h`

---

## **performDefaultImplementation**

*Overridden by subclasses to provide a default implementation for the command represented by the receiver.*

- (id)performDefaultImplementation

### Discussion

Do not invoke this method directly. [executeCommand](#) (page 12) invokes this method when the command being executed is not supported by the class of the objects receiving the command. The default implementation returns `nil`.

You need to create a subclass of `NSScriptCommand` only if you need to provide a default implementation of a command.

### Availability

Available in OS X v10.0 and later.

### Declared in

`NSScriptCommand.h`

---

## receiversSpecifier

---

*Returns the object specifier that, when evaluated, yields the receiver or receivers of the command.*

– (NSScriptObjectSpecifier \*)receiversSpecifier

### Discussion

The receiver is typically a container. For example, if the original command is `get the third paragraph of the first document`, the receiver specifier is `the first document`—it's the document that knows how to get or set words or paragraphs it contains.

### Availability

Available in OS X v10.0 and later.

### See Also

- [evaluatedReceivers](#) (page 11)
- [setReceiversSpecifier:](#) (page 18)

### Declared in

`NSScriptCommand.h`

---

## resumeExecutionWithResult:

---

*If a successful, unmatched, invocation of [suspendExecution](#) (page 22) has been made, resume the execution of the command.*

– (void)resumeExecutionWithResult:(id)result

### Discussion

Resumes the execution of the command if a successful, unmatched, invocation of [suspendExecution](#) (page 22) has been made—otherwise, does nothing. The value for `result` is dependent on the segment of command execution that was suspended:

- If `suspendExecution` was invoked from within a command handler of one of the command's receivers, `result` is considered to be the return value of the handler. Unless the command has received a [setScriptErrorNumber:](#) (page 20) message with a nonzero error number, execution of the command will continue and the command handlers of other receivers will be invoked.
- If `suspendExecution` was invoked from within an override of [performDefaultImplementation](#) (page 13) the result is treated as if it were the return value of the invocation of `performDefaultImplementation`.

`resumeExecutionWithResult:` may be invoked in any thread, not just the one in which the corresponding invocation of [suspendExecution](#) (page 22) occurred.

**Important:** The script command handler that is being executed when `suspendExecution` is invoked must return before you invoke `resumeExecutionWithResult:`. That is, it is not valid to suspend a command's execution and then resume it immediately.

### Availability

Available in OS X v10.3 and later.

### Declared in

`NSScriptCommand.h`

---

## [scriptErrorExpectedTypeDescriptor](#)

*Returns the type descriptor that was put in the reply Apple event if the sender requested a reply, execution of the receiver completed, and an error number was set.*

– (NSAppleEventDescriptor \*)scriptErrorExpectedTypeDescriptor

### Return Value

A descriptor that specifies a type.

### Discussion

When an error occurs during script command execution because an Apple event descriptor wasn't of the expected type, and the sender requested a reply, Cocoa scripting returns a descriptor for the expected type in a reply Apple event. You can invoke [setScriptErrorExpectedTypeDescriptor:](#) (page 19) to set this descriptor directly.

### Availability

Available in OS X v10.5 and later.

### Declared in

NSScriptCommand.h

## scriptErrorNumber

---

Returns the script error number, if any, associated with execution of the command.

– (int)scriptErrorNumber

### Discussion

When you subclass NSScriptCommand or one of its subclasses, you shouldn't need to override this method.

For error conditions specific to your application you can define your own error return values. For some common errors, you may want to return error values defined in `MacErrors.h`, a header in `CarbonCore.framework` (a subframework of `CoreServices.framework`). Look for error constants that start with `errAE`. For example, `errAEEventNotHandled` indicates a handler wasn't able to handle the Apple event.

### Availability

Available in OS X v10.0 and later.

### See Also

– [setScriptErrorNumber:](#) (page 20)

### Declared in

NSScriptCommand.h

## scriptErrorOffendingObjectDescriptor

---

Returns the object descriptor that was put in the reply Apple event if the sender requested a reply, execution of the receiver completed, and an error number was set.

– (NSAppleEventDescriptor \*)scriptErrorOffendingObjectDescriptor

### Return Value

A descriptor that specifies an object.

### Discussion

When an error that occurs during script command execution is caused by a specific object, and the sender requested a reply, Cocoa scripting returns a descriptor for the offending object in a reply Apple event. You can invoke [setScriptErrorOffendingObjectDescriptor:](#) (page 20) to set this descriptor directly.

### Availability

Available in OS X v10.5 and later.

### See Also

– [setScriptErrorOffendingObjectDescriptor:](#) (page 20)

### Declared in

NSScriptCommand.h

---

## scriptErrorString

---

*Returns the script error string, if any, associated with execution of the command.*

– (NSString \*)scriptErrorString

### Discussion

When you subclass NSScriptCommand or one of its subclasses, you shouldn't need to override this method.

### Availability

Available in OS X v10.0 and later.

### See Also

– [setScriptErrorString:](#) (page 21)

### Declared in

NSScriptCommand.h

---

## setArguments:

---

*Sets the arguments of the command to args.*

– (void)setArguments:(NSDictionary \*)args

### Discussion

Each argument in the dictionary is identified by the same name key used for the argument in the command's class declaration in the script suite file.

### Availability

Available in OS X v10.0 and later.

### See Also

- [arguments](#) (page 9)
- [evaluatedArguments](#) (page 11)

### Declared in

NSScriptCommand.h

### setDirectParameter:

---

*Sets the object that corresponds to the direct parameter of the Apple event from which the receiver derives.*

– (void)setDirectParameter:(id)directParameter

### Parameters

directParameter

An object to be set as the direct parameter.

### Discussion

You don't normally override this method.

### Availability

Available in OS X v10.0 and later.

### See Also

- [directParameter](#) (page 10)

### Declared in

NSScriptCommand.h

### setReceiversSpecifier:

---

*Sets the object specifier to receiversSpec that, when evaluated, indicates the receiver or receivers of the command.*

– (void)setReceiversSpecifier:(NSScriptObjectSpecifier \*)receiversSpec

## Discussion

If you create a subclass of `NSScriptCommand`, you don't necessarily need to override this method, though some of Cocoa's subclasses do. An override should perform the same function as the superclass method, with a critical difference: it causes the container specifier part of the passed-in object specifier to become the receiver specifier of the command, and the key part of the passed-in object specifier to become the key specifier. In an override, for example, if `receiversRef` is a specifier for the third rectangle of the first document, the receiver specifier is the first document while the key specifier is the third rectangle.

## Availability

Available in OS X v10.0 and later.

## See Also

- [evaluatedReceivers](#) (page 11)
- [receiversSpecifier](#) (page 14)

## Declared in

`NSScriptCommand.h`

---

## **setScriptErrorExpectedTypeDescriptor:**

*Sets a descriptor for the expected type that will be put in the reply Apple event if the sender requested a reply, execution of the receiver completes, and an error number was set.*

– (void)setScriptErrorExpectedTypeDescriptor:(NSAppleEventDescriptor \*)errorExpectedTypeDescriptor

## Parameters

`errorExpectedTypeDescriptor`  
A descriptor that specifies a type.

## Availability

Available in OS X v10.5 and later.

## See Also

- [scriptErrorExpectedTypeDescriptor](#) (page 15)

## Declared in

`NSScriptCommand.h`

## setScriptErrorNumber:

---

Sets a script error number that is associated with the execution of the command and is returned in the reply Apple event, if a reply was requested by the sender.

– (void)setScriptErrorNumber:(int)errorNumber

### Parameters

errorNumber

An error number to associate with the command.

### Discussion

If you override [performDefaultImplementation](#) (page 13) and an error occurs, you should call this method to supply an appropriate error number. In fact, any script handler should call this method when an error occurs. The error number you supply is returned in the reply Apple event.

Invoking `setScriptErrorNumber:` causes an error message to be displayed. To associate a specific error message with the error number, you invoke [setScriptErrorString:](#) (page 21). This makes sense, for example, when you set an error number that is specific to your application, or when you can supply a specific and useful error message to the user.

If `setScriptErrorNumber:` is invoked on an `NSScriptCommand` with multiple receivers, the command will stop sending command handling messages to more receivers.

### Availability

Available in OS X v10.0 and later.

### See Also

– [scriptErrorNumber](#) (page 16)

Related Sample Code  
Sketch

Sketch+Accessibility

### Declared in

`NSScriptCommand.h`

## setScriptErrorOffendingObjectDescriptor:

---

Sets a descriptor for an object that will be put in the reply Apple event if the sender requested a reply, execution of the receiver completes, and an error number was set.

– (void)setScriptErrorOffendingObjectDescriptor:(NSAppleEventDescriptor \*)errorOffendingObjectDescriptor

#### Parameters

errorOffendingObjectDescriptor

A descriptor that specifies an object that was responsible for an error.

#### Availability

Available in OS X v10.5 and later.

#### See Also

– [scriptErrorOffendingObjectDescriptor](#) (page 16)

#### Declared in

NSScriptCommand.h

### **setScriptErrorString:**

---

*Sets a script error string that is associated with execution of the command.*

– (void)setScriptErrorString:(NSString \*)errorString

#### Parameters

errorString

A string that describes an error.

#### Discussion

If you override [performDefaultImplementation](#) (page 13) and an error occurs, you should call this method to supply a string that provides a useful explanation. In fact, any script handler should call this method when an error occurs.

Calling this method alone does not cause an error message to be displayed—you must also call [setScriptErrorNumber:](#) (page 20) to supply an error number.

#### Availability

Available in OS X v10.0 and later.

#### See Also

– [scriptErrorString](#) (page 17)

#### Related Sample Code

Sketch

Sketch+Accessibility

**Declared in**  
NSScriptCommand.h

## suspendExecution

---

*Suspends the execution of the receiver.*

– (void)suspendExecution

### Discussion

Suspends the execution of the receiver only if the receiver is being executed in the current thread by Cocoa scripting's built-in Apple event handling (that is, the receiver would be returned by [NSScriptCommand currentCommand])—otherwise, does nothing. A matching invocation of [resumeExecutionWithResult:](#) (page 14) must be made.

**Important:** The script command handler that is being executed when this method is invoked must return before the subsequent invocation of [resumeExecutionWithResult:](#) (page 14). That is, it is not valid to suspend a command's execution and then resume it immediately.

Another command can execute while a command is suspended.

### Availability

Available in OS X v10.3 and later.

**Declared in**  
NSScriptCommand.h

## Constants

### NSScriptCommand—General Command Execution Errors

---

*NSScriptCommand uses the following error codes for general command execution problems:*

```
enum {
    NSNoScriptError = 0,
    NSReceiverEvaluationScriptError,
    NSKeySpecifierEvaluationScriptError,
    NSArgumentEvaluationScriptError,
    NSReceiversCantHandleCommandScriptError,
    NSRequiredArgumentsMissingScriptError,
```

```
    NSArgumentsWrongScriptError,  
    NSUnknownKeyScriptError,  
    NSInternalScriptError,  
    NSOperationNotSupportedForKeyScriptError,  
    NSCannotCreateScriptCommandError  
};
```

### Constants

#### NSNoScriptError

No error.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### NSReceiverEvaluationScriptError

The object or objects specified by the direct parameter to a command could not be found.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### NSKeySpecifierEvaluationScriptError

The object or objects specified by a key (for commands that support key specifiers) could not be found.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### NSArgumentEvaluationScriptError

The object specified by an argument could not be found.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### NSReceiversCantHandleCommandScriptError

The receivers don't support the command sent to them.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### NSRequiredArgumentsMissingScriptError

An argument (or more than one argument) is missing.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### NSArgumentsWrongScriptError

An argument (or more than one argument) is of the wrong type or is otherwise invalid.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### NSUnknownKeyScriptError

An unidentified error occurred; indicates an error in the scripting support of your application.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### NSInternalScriptError

An unidentified internal error occurred; indicates an error in the scripting support of your application.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### NSOperationNotSupportedForKeyScriptError

The implementation of a scripting command signaled an error.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### NSCannotCreateScriptCommandError

Could not create the script command; an invalid or unrecognized Apple event was received.

Available in OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

#### **Declared in**

`NSScriptCommand.h`

# Document Revision History

This table describes the changes to *NSScriptCommand Class Reference*.

Date	Notes
2007-07-20	<p>Added new methods for OS X version 10.5.</p> <p>The new methods are <a href="#">scriptErrorExpectedTypeDescriptor</a> (page 15), <a href="#">scriptErrorOffendingObjectDescriptor</a> (page 16), <a href="#">setScriptErrorExpectedTypeDescriptor:</a> (page 19), and <a href="#">setScriptErrorOffendingObjectDescriptor:</a> (page 20).</p> <p>Clarified the descriptions for <a href="#">setScriptErrorNumber:</a> (page 20) and <a href="#">setScriptErrorString:</a> (page 21).</p>
2006-05-23	First publication of this content as a separate document.



Apple Inc.  
Copyright © 2007 Apple Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Apple Inc.  
1 Infinite Loop  
Cupertino, CA 95014  
408-996-1010

Apple, the Apple logo, AppleScript, Carbon, Cocoa, Mac, and OS X are trademarks of Apple Inc., registered in the U.S. and other countries.

**Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.**

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.**

**Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.**